Lab4_students1

January 3, 2019

1 Data import

1.1 Question 0 - Get common wikidata occupations

Write a sparql query that retrieves the top 100 occupations on wikidata (wikidata property P106).

You may use the interface https://query.wikidata.org/ to try different queries. Here are some example sparql queries: https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples

```
In [2]: query = """
        SELECT ?o (COUNT(?person) AS ?count) WHERE
        {
           ?person wdt:P106 ?o
        }
        GROUP BY ?o
        ORDER BY DESC(?count)
        LIMIT 100
        """
```

The following assertion should pass if your answer is correct.

```
In [3]: import requests

        occupations = ['Q82955', 'Q937857', 'Q36180', 'Q33999', 'Q1650915', 'Q1028181', 'Q1930

        def evalSparql(query):
            return requests.post('https://query.wikidata.org/sparql', data=query, headers={
                'content-type': 'application/sparql-query',
                'accept': 'application/json',
                'user-agent': 'User:Tpt'
            }).json()['results']['bindings']

        myOccupations = [val['o']['value'].replace('http://www.wikidata.org/entity/', '')
                         for val in evalSparql(query)]
        assert(frozenset(occupations) == frozenset(myOccupations))
```

1.2 Occupations labels

We load the labels of the occupations from Wikidata

```
In [4]: occupations_label = {}

        query = """
        SELECT DISTINCT ?o ?oLabel
        WHERE {
            VALUES ?o { %s }
            SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
        }"""% ' '.join('wd:' + o for o in occupations)

        for result in evalSparql(query):
            occupations_label[result['o']['value'].replace('http://www.wikidata.org/entity/',

        print(occupations_label)
```

```
{'Q121594': 'professor', 'Q82955': 'politician', 'Q81096': 'engineer', 'Q177220': 'singer', 'Q
```

We load *all* the labels of the occupations from Wikipedia

```
In [5]: occupations_labels = {k: [v] for k, v in occupations_label.items()}

        query = """
        SELECT ?o ?altLabel
        WHERE {
          VALUES ?o { %s }
          ?o skos:altLabel ?altLabel . FILTER (lang(?altLabel) = "en")
        }""" % ' '.join('wd:' + o for o in occupations)

        for result in evalSparql(query):
            occupations_labels[result['o']['value'].replace('http://www.wikidata.org/entity/',

        print(occupations_labels)
```

```
{'Q121594': ['professor', 'Prof.'], 'Q82955': ['politician', 'political leader', 'polit.', 'po
```

1.3 Wikipedia articles

Here we load the training and the testing sets. To save memory space we use a generator that will read the file each time we iterate over the training or the testing examples.

```
In [6]: import gzip
        import json

        def loadJson(filename):
```

```
    with gzip.open(filename, 'rt') as fp:
        for line in fp:
            yield json.loads(line)

class MakeIter(object):
    def __init__(self, generator_func, **kwargs):
        self.generator_func = generator_func
        self.kwargs = kwargs
    def __iter__(self):
        return self.generator_func(**self.kwargs)

training_set = MakeIter(loadJson, filename='wiki-train.json.gz')
testing_set = MakeIter(loadJson, filename='wiki-test.json.gz')
```

2 Extract occupations from summaries

2.1 Task 1 - Dictionnary extraction

Using `occupations_labels` dictionnary, identify all occupations for each articles. Complete the function below to evaluate the accuracy of such approach. It will serve as a baseline.

In [7]:
```
label_to_occ = dict()
for key, occs in occupations_labels.items():
    for occ in occs:
        label_to_occ[occ.lower()] = key

def predict_dictionnary(example, occupations_labels):
    occs = []
    summary = example['summary'].lower()
    labels = label_to_occ.keys()
    for label in labels:
        if label in summary:
            occs.append(label_to_occ[label])
    return occs

def evaluate_dictionnary(training_set, occupations_labels):
    nexample = 0
    accuracy = 0.
    prediction = None
    for example in training_set:
        prediction = predict_dictionnary(example, occupations_labels)
        p = frozenset(prediction)
        g = frozenset(example['occupations'])
        accuracy += 1.*len(p & g) / len(p | g)
        nexample += 1
    return accuracy / nexample
```

```
        evaluate_dictionnary(training_set, occupations_labels)
```

```
Out[7]: 0.4842586814146957
```

2.2 Task 2 - Simple neural network

We load the articles "summary" and we take the average of the word vectors. This is done with spacy loaded with the fast text vectors. To do the installation/loading [takes 8-10 minutes, dl 1.2Go]

```
pip3 install spacy
wget https://s3-us-west-1.amazonaws.com/fasttext-vectors/cc.en.300.vec.gz
python3 -m spacy init-model en /tmp/en_vectors_wiki_lg --vectors-loc cc.en.300.vec.gz
rm cc.en.300.vec.gz
```

```
In [ ]: import spacy
        from sklearn.model_selection import train_test_split

        nlp = spacy.load('/tmp/en_vectors_wiki_lg')

        def vectorize(dataset, nlp):
            result = {}
            for example in dataset:
                doc = nlp(example['summary'], disable=['parser', 'tagger'])
                result[example['title']] = {}
                result[example['title']]['vector'] = doc.vector
                result[example['title']]['summary'] = example['summary']
                if 'occupations' in example:
                    result[example['title']]['occupations'] = example['occupations']
            return result

        vectorized_training = vectorize(training_set, nlp)
        vectorized_testing = vectorize(testing_set, nlp)
        nlp = None
```

```
In [10]: len(vectorized_training)
```

```
Out[10]: 427798
```

```
In [11]: v = vectorized_training['George_Washington']['vector']
         print(v)
```

```
[-1.45162819e-02 -2.45802402e-02 -4.59302496e-03 -4.09372151e-02
 -4.47662771e-02 -4.18604538e-03 -3.15232435e-03 -1.44802360e-02
 -1.68499984e-02 -3.69651243e-03 -1.16255814e-02  1.43651171e-02
  2.02674349e-03 -5.88953542e-03 -2.17011590e-02  1.02302311e-02
 -2.49313917e-02 -5.65232616e-03 -2.25581434e-02  8.29069968e-03
 -1.44069805e-03  2.25197673e-02 -6.81395701e-04 -1.37232570e-02
```

-1.26674427e-02	-3.35569866e-02	1.10627888e-02	-2.37208814e-03
-2.30000000e-02	7.58616179e-02	-5.03487710e-04	-2.51116175e-02
9.26511642e-03	-2.52558179e-02	-1.51058156e-02	-9.51627828e-03
1.17523270e-02	1.22441910e-03	1.08139520e-03	3.39302444e-03
2.20116391e-03	1.46860480e-02	-1.43686021e-02	5.76395402e-03
1.74162779e-02	-4.76220921e-02	-1.72569733e-02	-1.49988411e-02
-1.77732538e-02	1.58907007e-02	-7.23255938e-03	2.43825577e-02
-2.73104683e-02	-3.67430188e-02	-1.48802334e-02	-1.34825567e-02
-3.14348824e-02	1.95930228e-02	-6.68605033e-04	-9.24302172e-03
1.56976283e-04	-1.65674444e-02	-1.30372085e-02	6.16298130e-05
-3.63139645e-03	2.74534873e-03	-1.62697677e-02	-4.70697694e-03
5.48139494e-03	4.39302297e-03	4.65523303e-02	2.29872130e-02
2.72058025e-02	-5.52790612e-03	2.19720937e-02	-4.41581383e-02
1.33255811e-03	1.20244222e-02	3.49267460e-02	3.76593024e-02
8.65232572e-03	-6.52325572e-03	-1.90407019e-02	1.03569757e-02
1.09301973e-03	-6.28488278e-03	3.98965068e-02	-3.81744131e-02
-1.35965087e-02	1.74023230e-02	-1.48686031e-02	5.78604685e-03
-8.59186146e-03	4.74418374e-03	1.54720917e-02	-6.42325589e-03
-1.58430226e-02	-2.98779178e-02	-1.54255824e-02	3.28209326e-02
2.43825577e-02	1.32907031e-03	1.80883706e-02	-2.72825565e-02
9.28488653e-03	-7.39418622e-03	-7.98023026e-03	1.84244160e-02
-9.45350039e-04	-1.16825579e-02	1.15813862e-03	-2.10464321e-04
-3.00813979e-03	4.75407019e-02	-8.32790602e-03	4.11511678e-03
-1.25604663e-02	8.92209262e-03	7.64534995e-03	-2.65965052e-02
6.58837147e-03	-1.12011610e-02	-9.68022924e-03	1.60023291e-02
1.61629519e-04	3.20906974e-02	-1.59848798e-02	1.14162825e-02
-2.40430199e-02	5.39906919e-02	-4.80814092e-03	3.02209193e-03
5.89418598e-03	-3.94418649e-03	-2.68058274e-02	-8.98256153e-03
-2.94616278e-02	3.90697829e-03	4.68255766e-03	3.96162830e-03
-2.68069748e-02	-2.68395394e-02	-9.76740339e-05	5.67557989e-03
4.43197712e-02	-1.38953477e-02	-3.69888335e-01	1.04639539e-02
1.55372089e-02	-1.35093015e-02	-8.09988379e-02	2.67802346e-02
2.21941881e-02	-7.86627829e-03	-1.00313956e-02	1.52511625e-02
1.45744160e-01	4.61395411e-03	7.26162829e-03	3.14453505e-02
-7.95465056e-03	-1.25395320e-02	6.95348764e-03	-2.48023286e-03
6.17325725e-03	1.26546472e-02	1.03558144e-02	-1.21616265e-02
-1.27907039e-03	-1.99348871e-02	-9.01860371e-03	4.25581448e-03
7.45790750e-02	1.02186035e-02	-9.93953645e-03	1.72848776e-02
-1.03779081e-02	1.46616297e-02	-3.75465187e-03	-2.26953458e-02
5.36046689e-04	6.64511696e-02	-2.53790785e-02	5.80627881e-02
-1.42732579e-02	9.22453254e-02	-1.12825576e-02	-2.51837187e-02
3.90697736e-03	5.96395321e-03	-3.02476659e-02	2.63883732e-02
-1.69488378e-02	7.39418576e-03	1.60662793e-02	-1.68313961e-02
-8.25814065e-03	-1.36965141e-02	7.30697624e-03	1.63453538e-02
-4.15407047e-02	1.05633713e-01	1.53325591e-02	6.63023209e-03
3.93279046e-02	-1.27697680e-02	-5.95697621e-03	-8.67441762e-03
1.58593040e-02	9.42093134e-03	-4.15697647e-03	1.34639572e-02
-4.10383604e-02	-2.82325619e-03	-2.43790708e-02	-4.02325485e-03

```
  1.65058132e-02  4.21395432e-03  1.25813941e-02  1.64744183e-02
 -2.81162816e-03  1.34813897e-02 -8.19302350e-03 -7.04767322e-03
  1.67139638e-02  1.43581396e-02  1.20023256e-02  4.96162800e-03
  1.76325571e-02 -7.07674446e-03 -4.24197726e-02 -2.34697610e-02
 -1.86058115e-02 -2.32790736e-03  2.98906974e-02  1.53604464e-03
  1.95941851e-02 -2.67104693e-02 -1.12453466e-02 -2.54534930e-03
 -4.29302268e-03  3.56558077e-02 -4.36046888e-04 -8.16406980e-02
  5.04779041e-01 -2.18813960e-02  1.15883695e-02  2.14848872e-02
  7.80581404e-03  1.55116236e-02 -1.11523261e-02  4.61628864e-04
  1.72918607e-02  1.43034859e-02  2.05546506e-02 -8.23488459e-03
 -3.16290706e-02 -4.83953534e-03 -1.82697661e-02  2.02907110e-03
 -3.51163093e-04  1.10220918e-02 -8.54755938e-02 -2.68255756e-03
  1.83174424e-02  1.91116314e-02 -4.73488262e-03 -8.08255840e-03
  1.37906978e-02 -7.76046468e-03 -2.82767452e-02 -2.99069774e-03
  1.06569799e-02 -5.99999772e-03  1.11883730e-02  4.28720983e-03
 -3.12255807e-02 -8.07186142e-02  8.59302282e-03 -8.11744668e-03
 -5.36279054e-03  1.87046509e-02 -1.10972092e-01 -3.07988375e-02
  9.47441999e-03 -1.03662787e-02  1.16337193e-02  3.22093032e-02
 -2.69790720e-02  2.25430205e-02 -1.49802361e-02 -1.05290683e-02
 -4.36534919e-02  6.34883530e-04 -2.83197612e-02 -1.37674408e-02
 -1.50220934e-02  1.30851150e-01 -1.22430259e-02  2.38767453e-02]
```

In [12]:
```python
v.shape
```

Out[12]: (300,)

2.3 Split the vectorized_training into train and test set

In [13]:
```python
def splitDict(d, percent):
    split_idx = int(len(d) * percent)
    d1 = dict(list(d.items())[: split_idx])
    d2 = dict(list(d.items())[split_idx:])

    return d1, d2

vectorized_training_test, vectorized_training_train = splitDict(vectorized_training, (
```

In [14]:
```python
len(vectorized_training_train)
```

Out[14]: 342239

In [15]:
```python
# We encode the data
import numpy as np

def encode_data(vectorized_data):
    X = np.array([vectorized_data[article]['vector'] for article in vectorized_data])
    y = np.array([[(1 if occupation in vectorized_data[article]['occupations'] else 0
                        for occupation in occupations ] for article in vectorized_data
```

```
        return X, y

    X_train, y_train = encode_data(vectorized_training_train)
    X_test, y_test = encode_data(vectorized_training_test)
```

```
In [16]: print(len(y_train[0]))
```

```
100
```

```
In [17]: X_train.shape
```

```
Out[17]: (342239, 300)
```

```
In [18]: y_train.shape
```

```
Out[18]: (342239, 100)
```

> Using keras, define a sequential neural network with two layers. Use categorical_crossentropy as a loss function and softmax as the activation function of the output layer

You can look into the documentation here: https://keras.io/getting-started/sequential-model-guide/

```
In [19]: from keras.models import Sequential
         from keras.layers import Dense, Activation
         from keras.optimizers import Adam


         model = Sequential()
         model.add(Dense(512, activation='relu', input_dim=300))
         model.add(Dense(256, activation='relu'))
         model.add(Dense(100, activation='softmax'))

         optimizer = Adam()
         model.compile(optimizer=optimizer,
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
```

```
Using TensorFlow backend.
```

```
In [20]: history = model.fit(X_train, y_train, epochs=50, batch_size=1024, validation_split=0.
```

```
Train on 308015 samples, validate on 34224 samples
Epoch 1/50
308015/308015 [==============================] - 3s 10us/step - loss: 3.5608 - acc: 0.4881 - va
Epoch 2/50
308015/308015 [==============================] - 1s 5us/step - loss: 2.3868 - acc: 0.6581 - val
```

```
Epoch 3/50
308015/308015 [==============================] - 1s 4us/step - loss: 2.1847 - acc: 0.6880 - val
Epoch 4/50
308015/308015 [==============================] - 1s 4us/step - loss: 2.0853 - acc: 0.7033 - val
Epoch 5/50
308015/308015 [==============================] - 1s 4us/step - loss: 2.0255 - acc: 0.7115 - val
Epoch 6/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.9833 - acc: 0.7176 - val
Epoch 7/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.9517 - acc: 0.7213 - val
Epoch 8/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.9250 - acc: 0.7251 - val
Epoch 9/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.9036 - acc: 0.7273 - val
Epoch 10/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.8846 - acc: 0.7293 - val
Epoch 11/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.8676 - acc: 0.7324 - val
Epoch 12/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.8522 - acc: 0.7333 - val
Epoch 13/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.8387 - acc: 0.7349 - val
Epoch 14/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.8256 - acc: 0.7372 - val
Epoch 15/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.8138 - acc: 0.7383 - val
Epoch 16/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.8025 - acc: 0.7391 - val
Epoch 17/50
308015/308015 [==============================] - 1s 5us/step - loss: 1.7916 - acc: 0.7408 - val
Epoch 18/50
308015/308015 [==============================] - 1s 5us/step - loss: 1.7818 - acc: 0.7417 - val
Epoch 19/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.7721 - acc: 0.7434 - val
Epoch 20/50
308015/308015 [==============================] - 1s 5us/step - loss: 1.7634 - acc: 0.7436 - val
Epoch 21/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.7554 - acc: 0.7450 - val
Epoch 22/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.7462 - acc: 0.7463 - val
Epoch 23/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.7388 - acc: 0.7467 - val
Epoch 24/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.7308 - acc: 0.7481 - val
Epoch 25/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.7240 - acc: 0.7486 - val
Epoch 26/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.7173 - acc: 0.7494 - val
```

```
Epoch 27/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.7097 - acc: 0.7500 - va
Epoch 28/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.7030 - acc: 0.7510 - va
Epoch 29/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6968 - acc: 0.7518 - va
Epoch 30/50
308015/308015 [==============================] - 1s 5us/step - loss: 1.6907 - acc: 0.7525 - va
Epoch 31/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6837 - acc: 0.7533 - va
Epoch 32/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6777 - acc: 0.7537 - va
Epoch 33/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6719 - acc: 0.7541 - va
Epoch 34/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6654 - acc: 0.7554 - va
Epoch 35/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6594 - acc: 0.7557 - va
Epoch 36/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6540 - acc: 0.7566 - va
Epoch 37/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6487 - acc: 0.7568 - va
Epoch 38/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6427 - acc: 0.7581 - va
Epoch 39/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6375 - acc: 0.7579 - va
Epoch 40/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6323 - acc: 0.7581 - va
Epoch 41/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6270 - acc: 0.7595 - va
Epoch 42/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6214 - acc: 0.7601 - va
Epoch 43/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6162 - acc: 0.7604 - va
Epoch 44/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6115 - acc: 0.7606 - va
Epoch 45/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6061 - acc: 0.7616 - va
Epoch 46/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.6012 - acc: 0.7622 - va
Epoch 47/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.5964 - acc: 0.7628 - va
Epoch 48/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.5910 - acc: 0.7632 - va
Epoch 49/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.5868 - acc: 0.7636 - va
Epoch 50/50
308015/308015 [==============================] - 1s 4us/step - loss: 1.5814 - acc: 0.7645 - va
```

Complete the function predict: output the list of occupations where the corresponding neuron on the output layer of our model has a value > 0.1

```
In [25]: def predict_nn(model, article_name, vectorized_dataset):
             input_vector = vectorized_dataset[article_name]['vector'].reshape((1, 300))
             scores = model.predict(input_vector).reshape(100)
             predictions = np.where(scores > 0.1)[0]
         #     print(scores[predictions])
             return set(np.array(occupations)[predictions])

         print(predict_nn(model, 'Elvis_Presley', vectorized_training))
         # should be {'Q177220'}
```

```
{'Q177220', 'Q639669', 'Q33999'}
```

```
In [22]: def evaluate_nn(vectorized_training, model):
             nexample = 0
             accuracy = 0.
             prediction = None
             for article_name in vectorized_training:
                 prediction = predict_nn(model, article_name, vectorized_training)
                 p = frozenset(prediction)
                 g = frozenset(vectorized_training[article_name]['occupations'])
                 accuracy += 1.*len(p & g) / len(p | g)
                 nexample += 1
             return accuracy / nexample
```

```
In [23]: print(evaluate_nn(vectorized_training_train, model))
         print(evaluate_nn(vectorized_training_test, model))
```

```
0.7048576643356244
0.6662116943899452
```

2.4 Task 3 Your approach: CNN + BiRNN

```
In [7]: from keras.preprocessing.text import Tokenizer
        from keras.preprocessing.sequence import pad_sequences
        import numpy as np
        import os
        from keras.models import Sequential
        from keras.layers import Embedding, Flatten, Dense, GRU, Dropout, Conv1D, MaxPooling1D
        from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
```

```
Using TensorFlow backend.
```

```
In [8]: # Extract the dataset into summaries, titles and occupations
        def parse(dataset):
            titles = []
            summaries = []
            occs = []
            for example in dataset:
                titles.append(example['title'])
                summaries.append(example['summary'])
                if 'occupations' in example:
                    occs.append(example['occupations'])
                else:
                    occs.append([])
            return titles, summaries, occs

        titles_train, summaries_train, occs_train = parse(training_set)

        s = int(len(titles_train) * 0.8)
        titles_train_train, summaries_train_train, occs_train_train = titles_train[:s], summar
        titles_train_test, summaries_train_test, occs_train_test = titles_train[s:], summaries_

        titles_test, summaries_test, occs_test = parse(testing_set)
```

```
In [9]: n_samples = len(titles_train_train)
        maxlen = 300
        training_samples = int(n_samples * 0.85)
        validation_samples = n_samples - training_samples
        max_words = 20000

        tokenizer = Tokenizer(num_words=max_words)
        tokenizer.fit_on_texts(summaries_train_train)

        # convert text to sequences
        sequences =  tokenizer.texts_to_sequences(summaries_train_train)
        sequences_test = tokenizer.texts_to_sequences(summaries_train_test)

        word_index = tokenizer.word_index
        print('Found', len(word_index), 'unique tokens.')
```

```
Found 370295 unique tokens.
```

```
In [10]: def convert_occs_to_labels(occupations, occs_train):
             labels = []
             for i in range(len(occs_train)):
                 label = []
                 for occ in occupations:
                     if occ in occs_train[i]:
                         label.append(1)
```

```
                else:
                    label.append(0)
            labels.append(label)
        return np.array(labels)
```

In [11]:
```
data = pad_sequences(sequences, maxlen=maxlen)
data_test = pad_sequences(sequences_test, maxlen=maxlen)
labels = convert_occs_to_labels(occupations, occs_train_train)

print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

# shuffle the data
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

# split into training and testing set
x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Shape of data tensor: (342333, 300)
Shape of label tensor: (342333, 100)
```

In [12]:
```
glove_dir = 'glove.6B'
embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.300d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Found', len(embeddings_index), 'word vectors.')
```

```
Found 400000 word vectors.
```

In [13]:
```
# Build embedding matrix to load into embedding layer
embedding_dim = 300
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

In [22]:
```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))

model.add(Conv1D(64,kernel_size=3,padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=3))
model.add(Dropout(0.15))

model.add(Conv1D(128,kernel_size=3,padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=3))
model.add(Dropout(0.15))

model.add(Conv1D(256,kernel_size=3,padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=3))
model.add(Dropout(0.15))


model.add(Bidirectional(GRU(200, return_sequences=True, recurrent_dropout = 0.15)))

model.add(Bidirectional(GRU(150, return_sequences=True, recurrent_dropout = 0.15)))

model.add(Bidirectional(GRU(150, recurrent_dropout = 0.1)))

model.add(Dense(512,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.15))

model.add(Dense(256,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.15))

model.add(Dense(256,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.15))


model.add(Dense(100, activation='sigmoid'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 300, 300)          6000000
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 300, 64)           57664
_________________________________________________________________
```

```
batch_normalization_13 (Batc (None, 300, 64)          256
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 100, 64)          0
_________________________________________________________________
dropout_13 (Dropout)         (None, 100, 64)          0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 100, 128)         24704
_________________________________________________________________
batch_normalization_14 (Batc (None, 100, 128)         512
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 33, 128)          0
_________________________________________________________________
dropout_14 (Dropout)         (None, 33, 128)          0
_________________________________________________________________
conv1d_9 (Conv1D)            (None, 33, 256)          98560
_________________________________________________________________
batch_normalization_15 (Batc (None, 33, 256)          1024
_________________________________________________________________
max_pooling1d_9 (MaxPooling1 (None, 11, 256)          0
_________________________________________________________________
dropout_15 (Dropout)         (None, 11, 256)          0
_________________________________________________________________
bidirectional_8 (Bidirection (None, 11, 400)          548400
_________________________________________________________________
bidirectional_9 (Bidirection (None, 11, 300)          495900
_________________________________________________________________
bidirectional_10 (Bidirectio (None, 300)              405900
_________________________________________________________________
dense_9 (Dense)              (None, 512)              154112
_________________________________________________________________
batch_normalization_16 (Batc (None, 512)              2048
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)              0
_________________________________________________________________
dense_10 (Dense)             (None, 256)              131328
_________________________________________________________________
batch_normalization_17 (Batc (None, 256)              1024
_________________________________________________________________
dropout_17 (Dropout)         (None, 256)              0
_________________________________________________________________
dense_11 (Dense)             (None, 256)              65792
_________________________________________________________________
batch_normalization_18 (Batc (None, 256)              1024
_________________________________________________________________
dropout_18 (Dropout)         (None, 256)              0
_________________________________________________________________
dense_12 (Dense)             (None, 100)              25700
=================================================================
```

```
Total params: 8,013,948
Trainable params: 8,011,004
Non-trainable params: 2,944
_________________________________________________________________
```

```
In [23]: # Load the Glove embedding in the model
         model.layers[0].set_weights([embedding_matrix])

         # we will not update this layer during training.
         # If I trained this layer, the test accuracy would decrease
         model.layers[0].trainable = False
```

```
In [24]: import tensorflow as tf
         import keras.backend.tensorflow_backend as tfb

         POS_WEIGHT = 10  # Tested with other values (5, 15, 20, 100). 10 is optimal number

         def weighted_binary_crossentropy(target, output):
             """
             Weighted binary crossentropy between an output tensor
             and a target tensor. POS_WEIGHT is used as a multiplier
             for the positive targets.

             Combination of the following functions:
             * keras.losses.binary_crossentropy
             * keras.backend.tensorflow_backend.binary_crossentropy
             * tf.nn.weighted_cross_entropy_with_logits
             """
             # transform back to logits
             _epsilon = tfb._to_tensor(tfb.epsilon(), output.dtype.base_dtype)
             output = tf.clip_by_value(output, _epsilon, 1 - _epsilon)
             output = tf.log(output / (1 - output))
             # compute weighted loss
             loss = tf.nn.weighted_cross_entropy_with_logits(targets=target,
                                                             logits=output,
                                                             pos_weight=POS_WEIGHT)
             return tf.reduce_mean(loss, axis=-1)
```

```
In [25]: callbacks = [
             ReduceLROnPlateau(monitor='val_loss',
                               factor=0.2,
                               patience=5,
                               verbose=1,
                               mode='auto',
                               min_delta=0.0001,
                               cooldown=0,
                               min_lr=0),
```

```
        ModelCheckpoint(filepath='best_model.h5', monitor='val_loss', save_best_only=True
    ]

    model.compile(optimizer='rmsprop',
        loss=weighted_binary_crossentropy,
        metrics=['acc'])

    history = model.fit(x_train, y_train,
        epochs=50,
        callbacks=callbacks,
        batch_size=2000,
        validation_data=(x_val, y_val))
```

```
Train on 290983 samples, validate on 51350 samples
Epoch 1/50
290983/290983 [==============================] - 86s 294us/step - loss: 0.4379 - acc: 0.4384 -

Epoch 00001: val_loss improved from inf to 0.16319, saving model to best_model.h5
Epoch 2/50
290983/290983 [==============================] - 81s 278us/step - loss: 0.1416 - acc: 0.6342 -

Epoch 00002: val_loss improved from 0.16319 to 0.12406, saving model to best_model.h5
Epoch 3/50
290983/290983 [==============================] - 79s 270us/step - loss: 0.1134 - acc: 0.6953 -

Epoch 00003: val_loss improved from 0.12406 to 0.10758, saving model to best_model.h5
Epoch 4/50
290983/290983 [==============================] - 79s 271us/step - loss: 0.1023 - acc: 0.7189 -

Epoch 00004: val_loss improved from 0.10758 to 0.09567, saving model to best_model.h5
Epoch 5/50
290983/290983 [==============================] - 79s 270us/step - loss: 0.0968 - acc: 0.7305 -

Epoch 00005: val_loss improved from 0.09567 to 0.09316, saving model to best_model.h5
Epoch 6/50
290983/290983 [==============================] - 79s 271us/step - loss: 0.0933 - acc: 0.7376 -

Epoch 00006: val_loss improved from 0.09316 to 0.09003, saving model to best_model.h5
Epoch 7/50
290983/290983 [==============================] - 82s 282us/step - loss: 0.0906 - acc: 0.7438 -

Epoch 00007: val_loss improved from 0.09003 to 0.08882, saving model to best_model.h5
Epoch 8/50
290983/290983 [==============================] - 81s 279us/step - loss: 0.0884 - acc: 0.7470 -

Epoch 00008: val_loss improved from 0.08882 to 0.08713, saving model to best_model.h5
Epoch 9/50
290983/290983 [==============================] - 79s 272us/step - loss: 0.0869 - acc: 0.7494 -
```

```
Epoch 00009: val_loss improved from 0.08713 to 0.08526, saving model to best_model.h5
Epoch 10/50
290983/290983 [==============================] - 79s 270us/step - loss: 0.0854 - acc: 0.7523 -

Epoch 00010: val_loss improved from 0.08526 to 0.08409, saving model to best_model.h5
Epoch 11/50
290983/290983 [==============================] - 82s 282us/step - loss: 0.0842 - acc: 0.7536 -

Epoch 00011: val_loss improved from 0.08409 to 0.08273, saving model to best_model.h5
Epoch 12/50
290983/290983 [==============================] - 79s 272us/step - loss: 0.0831 - acc: 0.7559 -

Epoch 00012: val_loss improved from 0.08273 to 0.08230, saving model to best_model.h5
Epoch 13/50
290983/290983 [==============================] - 80s 273us/step - loss: 0.0820 - acc: 0.7566 -

Epoch 00013: val_loss did not improve from 0.08230
Epoch 14/50
290983/290983 [==============================] - 79s 271us/step - loss: 0.0811 - acc: 0.7578 -

Epoch 00014: val_loss improved from 0.08230 to 0.08161, saving model to best_model.h5
Epoch 15/50
290983/290983 [==============================] - 80s 276us/step - loss: 0.0804 - acc: 0.7582 -

Epoch 00015: val_loss improved from 0.08161 to 0.08139, saving model to best_model.h5
Epoch 16/50
290983/290983 [==============================] - 80s 274us/step - loss: 0.0797 - acc: 0.7597 -

Epoch 00016: val_loss improved from 0.08139 to 0.08114, saving model to best_model.h5
Epoch 17/50
290983/290983 [==============================] - 79s 271us/step - loss: 0.0792 - acc: 0.7610 -

Epoch 00017: val_loss did not improve from 0.08114
Epoch 18/50
290983/290983 [==============================] - 79s 271us/step - loss: 0.0784 - acc: 0.7611 -

Epoch 00018: val_loss improved from 0.08114 to 0.07999, saving model to best_model.h5
Epoch 19/50
290983/290983 [==============================] - 79s 271us/step - loss: 0.0780 - acc: 0.7613 -

Epoch 00019: val_loss did not improve from 0.07999
Epoch 20/50
290983/290983 [==============================] - 81s 278us/step - loss: 0.0775 - acc: 0.7616 -

Epoch 00020: val_loss improved from 0.07999 to 0.07968, saving model to best_model.h5
Epoch 21/50
290983/290983 [==============================] - 79s 273us/step - loss: 0.0769 - acc: 0.7624 -
```

```
Epoch 00021: val_loss did not improve from 0.07968
Epoch 22/50
290983/290983 [==============================] - 80s 274us/step - loss: 0.0764 - acc: 0.7628 -

Epoch 00022: val_loss did not improve from 0.07968
Epoch 23/50
290983/290983 [==============================] - 80s 274us/step - loss: 0.0761 - acc: 0.7623 -

Epoch 00023: val_loss improved from 0.07968 to 0.07952, saving model to best_model.h5
Epoch 24/50
290983/290983 [==============================] - 80s 274us/step - loss: 0.0757 - acc: 0.7630 -

Epoch 00024: val_loss did not improve from 0.07952
Epoch 25/50
290983/290983 [==============================] - 79s 273us/step - loss: 0.0752 - acc: 0.7628 -

Epoch 00025: val_loss did not improve from 0.07952
Epoch 26/50
290983/290983 [==============================] - 83s 285us/step - loss: 0.0748 - acc: 0.7638 -

Epoch 00026: val_loss did not improve from 0.07952
Epoch 27/50
290983/290983 [==============================] - 86s 297us/step - loss: 0.0744 - acc: 0.7650 -

Epoch 00027: val_loss did not improve from 0.07952
Epoch 28/50
290983/290983 [==============================] - 90s 311us/step - loss: 0.0742 - acc: 0.7649 -

Epoch 00028: val_loss improved from 0.07952 to 0.07825, saving model to best_model.h5
Epoch 29/50
290983/290983 [==============================] - 87s 298us/step - loss: 0.0736 - acc: 0.7648 -

Epoch 00029: val_loss did not improve from 0.07825
Epoch 30/50
290983/290983 [==============================] - 87s 300us/step - loss: 0.0735 - acc: 0.7644 -

Epoch 00030: val_loss did not improve from 0.07825
Epoch 31/50
290983/290983 [==============================] - 90s 309us/step - loss: 0.0732 - acc: 0.7649 -

Epoch 00031: val_loss did not improve from 0.07825
Epoch 32/50
290983/290983 [==============================] - 88s 304us/step - loss: 0.0730 - acc: 0.7655 -

Epoch 00032: val_loss did not improve from 0.07825
Epoch 33/50
290983/290983 [==============================] - 85s 293us/step - loss: 0.0725 - acc: 0.7661 -
```

```
Epoch 00033: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.

Epoch 00033: val_loss did not improve from 0.07825
Epoch 34/50
290983/290983 [==============================] - 87s 299us/step - loss: 0.0700 - acc: 0.7690 -

Epoch 00034: val_loss improved from 0.07825 to 0.07715, saving model to best_model.h5
Epoch 35/50
290983/290983 [==============================] - 90s 309us/step - loss: 0.0693 - acc: 0.7697 -

Epoch 00035: val_loss did not improve from 0.07715
Epoch 36/50
290983/290983 [==============================] - 84s 289us/step - loss: 0.0688 - acc: 0.7694 -

Epoch 00036: val_loss did not improve from 0.07715
Epoch 37/50
290983/290983 [==============================] - 91s 312us/step - loss: 0.0684 - acc: 0.7695 -

Epoch 00037: val_loss did not improve from 0.07715
Epoch 38/50
290983/290983 [==============================] - 92s 316us/step - loss: 0.0682 - acc: 0.7698 -

Epoch 00038: val_loss did not improve from 0.07715
Epoch 39/50
290983/290983 [==============================] - 92s 318us/step - loss: 0.0680 - acc: 0.7699 -

Epoch 00039: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.

Epoch 00039: val_loss did not improve from 0.07715
Epoch 40/50
290983/290983 [==============================] - 83s 285us/step - loss: 0.0675 - acc: 0.7699 -

Epoch 00040: val_loss did not improve from 0.07715
Epoch 41/50
290983/290983 [==============================] - 82s 282us/step - loss: 0.0673 - acc: 0.7703 -

Epoch 00041: val_loss did not improve from 0.07715
Epoch 42/50
290983/290983 [==============================] - 81s 280us/step - loss: 0.0673 - acc: 0.7706 -

Epoch 00042: val_loss did not improve from 0.07715
Epoch 43/50
290983/290983 [==============================] - 86s 295us/step - loss: 0.0672 - acc: 0.7702 -

Epoch 00043: val_loss did not improve from 0.07715
Epoch 44/50
290983/290983 [==============================] - 89s 307us/step - loss: 0.0671 - acc: 0.7703 -
```

```
Epoch 00044: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.

Epoch 00044: val_loss did not improve from 0.07715
Epoch 45/50
290983/290983 [==============================] - 81s 279us/step - loss: 0.0670 - acc: 0.7712 -

Epoch 00045: val_loss did not improve from 0.07715
Epoch 46/50
290983/290983 [==============================] - 82s 283us/step - loss: 0.0670 - acc: 0.7706 -

Epoch 00046: val_loss did not improve from 0.07715
Epoch 47/50
290983/290983 [==============================] - 83s 284us/step - loss: 0.0670 - acc: 0.7707 -

Epoch 00047: val_loss did not improve from 0.07715
Epoch 48/50
290983/290983 [==============================] - 83s 284us/step - loss: 0.0669 - acc: 0.7707 -

Epoch 00048: val_loss did not improve from 0.07715
Epoch 49/50
290983/290983 [==============================] - 80s 276us/step - loss: 0.0669 - acc: 0.7711 -

Epoch 00049: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.

Epoch 00049: val_loss did not improve from 0.07715
Epoch 50/50
290983/290983 [==============================] - 91s 313us/step - loss: 0.0668 - acc: 0.7700 -

Epoch 00050: val_loss did not improve from 0.07715
```

In [ ]:
```
# Plot the results
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
```

```
        plt.plot(epochs, val_loss, 'b', label='Validation loss')
        plt.title('Training and validation loss')
        plt.legend()
        plt.show()
```

In [26]:
```
model.save_weights("model5.h5")
```

In [29]:
```
model.load_weights("best_model.h5")
```

In [27]:
```
def predict_nn_2(model, input_vector, print_score = False):

        scores = model.predict(input_vector).reshape(100)
        predictions1 = np.where(scores > 0.3)[0]
        predictions2 = np.where(scores > 0.5)[0]
        predictions3 = np.where(scores > 0.6)[0]
        predictions4 = np.where(scores > 0.7)[0]
        predictions5 = np.where(scores > 0.8)[0]
        predictions6 = np.where(scores > 0.9)[0]
        if print_score:
            print(scores[predictions1])
            print(scores[predictions2])
            print(scores[predictions3])
            print(scores[predictions4])
            print(scores[predictions5])
            print(scores[predictions6])
        res1 = set(np.array(occupations)[predictions1])
        res2 = set(np.array(occupations)[predictions2])
        res3 = set(np.array(occupations)[predictions3])
        res4 = set(np.array(occupations)[predictions4])
        res5 = set(np.array(occupations)[predictions5])
        res6 = set(np.array(occupations)[predictions6])
        return res1, res2, res3, res4, res5, res6
```

In [30]:
```
def evaluate_nn_2(titles, input_vectors, occs, model):
        nexample = len(titles)
        accuracy1 = 0.
        accuracy2 = 0.
        accuracy3 = 0.
        accuracy4 = 0.
        accuracy5 = 0.
        accuracy6 = 0.
        prediction = None
        for i in range(len(titles)):
            input_vector = input_vectors[i].reshape(1, -1)
            prediction1, prediction2, prediction3, prediction4, prediction5, prediction6 =
            p1 = frozenset(prediction1)
            p2 = frozenset(prediction2)
            p3 = frozenset(prediction3)
            p4 = frozenset(prediction4)
```

```
            p5 = frozenset(prediction5)
            p6 = frozenset(prediction6)
            g = frozenset(occs[i])
            accuracy1 += 1. / nexample * len(p1 & g) / len(p1 | g)
            accuracy2 += 1. / nexample * len(p2 & g) / len(p2 | g)
            accuracy3 += 1. / nexample * len(p3 & g) / len(p3 | g)
            accuracy4 += 1. / nexample * len(p4 & g) / len(p4 | g)
            accuracy5 += 1. / nexample * len(p5 & g) / len(p5 | g)
            accuracy6 += 1. / nexample * len(p5 & g) / len(p6 | g)
            if i % 1000 == 0:
                print("=========================")
                print(round(i / nexample, 4), " : ", round(accuracy1, 4))
                print(round(i / nexample, 4), " : ", round(accuracy2, 4))
                print(round(i / nexample, 4), " : ", round(accuracy3, 4))
                print(round(i / nexample, 4), " : ", round(accuracy4, 4))
                print(round(i / nexample, 4), " : ", round(accuracy5, 4))
                print(round(i / nexample, 4), " : ", round(accuracy6, 4))
        return accuracy1, accuracy2, accuracy3, accuracy4, accuracy5, accuracy6

    # print(evaluate_nn_2(titles_train, summaries_train, occs_train, model))
    print(evaluate_nn_2(titles_train_test, data_test, occs_train_test, model))
```

```
=========================
0.0  :  0.0
0.0  :  0.0
0.0  :  0.0
0.0  :  0.0
0.0  :  0.0
0.0  :  0.0
=========================
0.0117  :  0.0069
0.0117  :  0.0079
0.0117  :  0.0082
0.0117  :  0.0085
0.0117  :  0.0088
0.0117  :  0.0093
=========================
0.0234  :  0.0139
0.0234  :  0.0158
0.0234  :  0.0165
0.0234  :  0.0171
0.0234  :  0.0176
0.0234  :  0.0187
=========================
0.0351  :  0.0205
0.0351  :  0.0234
0.0351  :  0.0245
0.0351  :  0.0256
```

```
0.0351  :  0.0263
0.0351  :  0.028
=========================
0.0467  :  0.0274
0.0467  :  0.0312
0.0467  :  0.0327
0.0467  :  0.0342
0.0467  :  0.0352
0.0467  :  0.0375
=========================
0.0584  :  0.034
0.0584  :  0.0387
0.0584  :  0.0406
0.0584  :  0.0425
0.0584  :  0.0438
0.0584  :  0.0467
=========================
0.0701  :  0.0408
0.0701  :  0.0464
0.0701  :  0.0487
0.0701  :  0.0509
0.0701  :  0.0526
0.0701  :  0.0561
=========================
0.0818  :  0.048
0.0818  :  0.0546
0.0818  :  0.0572
0.0818  :  0.0598
0.0818  :  0.0617
0.0818  :  0.0657
=========================
0.0935  :  0.0547
0.0935  :  0.0623
0.0935  :  0.0653
0.0935  :  0.0683
0.0935  :  0.0704
0.0935  :  0.075
=========================
0.1052  :  0.0619
0.1052  :  0.0702
0.1052  :  0.0736
0.1052  :  0.0769
0.1052  :  0.0793
0.1052  :  0.0846
=========================
0.1168  :  0.0687
0.1168  :  0.0781
0.1168  :  0.0818
```

```
0.1168  :  0.0854
0.1168  :  0.088
0.1168  :  0.0938
=========================
0.1285  :  0.0758
0.1285  :  0.086
0.1285  :  0.0901
0.1285  :  0.094
0.1285  :  0.0968
0.1285  :  0.1032
=========================
0.1402  :  0.0827
0.1402  :  0.0938
0.1402  :  0.0983
0.1402  :  0.1026
0.1402  :  0.1059
0.1402  :  0.1127
=========================
0.1519  :  0.0895
0.1519  :  0.1016
0.1519  :  0.1065
0.1519  :  0.1111
0.1519  :  0.1147
0.1519  :  0.122
=========================
0.1636  :  0.0966
0.1636  :  0.1095
0.1636  :  0.1149
0.1636  :  0.1197
0.1636  :  0.1236
0.1636  :  0.1315
=========================
0.1753  :  0.1035
0.1753  :  0.1173
0.1753  :  0.1231
0.1753  :  0.1283
0.1753  :  0.1325
0.1753  :  0.1409
=========================
0.187  :  0.1107
0.187  :  0.1253
0.187  :  0.1315
0.187  :  0.137
0.187  :  0.1416
0.187  :  0.1505
=========================
0.1986  :  0.1175
0.1986  :  0.133
```

```
0.1986  :  0.1396
0.1986  :  0.1455
0.1986  :  0.1503
0.1986  :  0.1599
=========================
0.2103  :  0.1249
0.2103  :  0.1413
0.2103  :  0.1482
0.2103  :  0.1543
0.2103  :  0.1594
0.2103  :  0.1694
=========================
0.222  :  0.1313
0.222  :  0.1486
0.222  :  0.156
0.222  :  0.1624
0.222  :  0.1679
0.222  :  0.1786
=========================
0.2337  :  0.1386
0.2337  :  0.1568
0.2337  :  0.1644
0.2337  :  0.1713
0.2337  :  0.1771
0.2337  :  0.1882
=========================
0.2454  :  0.1458
0.2454  :  0.1649
0.2454  :  0.173
0.2454  :  0.1802
0.2454  :  0.1862
0.2454  :  0.1979
=========================
0.2571  :  0.1538
0.2571  :  0.1738
0.2571  :  0.1821
0.2571  :  0.1896
0.2571  :  0.1959
0.2571  :  0.208
=========================
0.2687  :  0.1608
0.2687  :  0.1816
0.2687  :  0.1903
0.2687  :  0.1981
0.2687  :  0.2048
0.2687  :  0.2176
=========================
0.2804  :  0.1681
```

```
0.2804  :  0.1897
0.2804  :  0.1987
0.2804  :  0.2069
0.2804  :  0.2139
0.2804  :  0.2272
=========================
0.2921  :  0.1755
0.2921  :  0.1978
0.2921  :  0.2072
0.2921  :  0.2155
0.2921  :  0.2228
0.2921  :  0.2367
=========================
0.3038  :  0.183
0.3038  :  0.206
0.3038  :  0.2157
0.3038  :  0.2244
0.3038  :  0.232
0.3038  :  0.2463
=========================
0.3155  :  0.1911
0.3155  :  0.2147
0.3155  :  0.2247
0.3155  :  0.2336
0.3155  :  0.2415
0.3155  :  0.2562
=========================
0.3272  :  0.1987
0.3272  :  0.223
0.3272  :  0.2333
0.3272  :  0.2425
0.3272  :  0.2507
0.3272  :  0.2659
=========================
0.3388  :  0.2065
0.3388  :  0.2314
0.3388  :  0.2419
0.3388  :  0.2514
0.3388  :  0.2598
0.3388  :  0.2756
=========================
0.3505  :  0.2139
0.3505  :  0.2395
0.3505  :  0.2504
0.3505  :  0.2602
0.3505  :  0.269
0.3505  :  0.2851
=========================
```

```
0.3622  :  0.2209
0.3622  :  0.2472
0.3622  :  0.2583
0.3622  :  0.2685
0.3622  :  0.2775
0.3622  :  0.2942
=========================
0.3739  :  0.2286
0.3739  :  0.2557
0.3739  :  0.267
0.3739  :  0.2775
0.3739  :  0.2868
0.3739  :  0.3039
=========================
0.3856  :  0.2355
0.3856  :  0.2632
0.3856  :  0.2749
0.3856  :  0.2857
0.3856  :  0.295
0.3856  :  0.3125
=========================
0.3973  :  0.2406
0.3973  :  0.2695
0.3973  :  0.2816
0.3973  :  0.2928
0.3973  :  0.3022
0.3973  :  0.3204
=========================
0.409  :  0.2467
0.409  :  0.2766
0.409  :  0.2891
0.409  :  0.3005
0.409  :  0.3101
0.409  :  0.3287
=========================
0.4206  :  0.2531
0.4206  :  0.2839
0.4206  :  0.2968
0.4206  :  0.3084
0.4206  :  0.3179
0.4206  :  0.337
=========================
0.4323  :  0.2597
0.4323  :  0.2914
0.4323  :  0.3047
0.4323  :  0.3165
0.4323  :  0.326
0.4323  :  0.3456
```

```
=========================
0.444  :  0.2665
0.444  :  0.299
0.444  :  0.3126
0.444  :  0.3247
0.444  :  0.3343
0.444  :  0.3542
=========================
0.4557  :  0.2732
0.4557  :  0.3067
0.4557  :  0.3206
0.4557  :  0.3329
0.4557  :  0.3427
0.4557  :  0.3632
=========================
0.4674  :  0.2799
0.4674  :  0.3142
0.4674  :  0.3285
0.4674  :  0.341
0.4674  :  0.351
0.4674  :  0.3719
=========================
0.4791  :  0.2866
0.4791  :  0.3219
0.4791  :  0.3366
0.4791  :  0.3493
0.4791  :  0.3594
0.4791  :  0.3808
=========================
0.4907  :  0.2932
0.4907  :  0.3293
0.4907  :  0.3444
0.4907  :  0.3574
0.4907  :  0.3676
0.4907  :  0.3895
=========================
0.5024  :  0.2998
0.5024  :  0.3368
0.5024  :  0.3522
0.5024  :  0.3655
0.5024  :  0.3759
0.5024  :  0.3984
=========================
0.5141  :  0.3067
0.5141  :  0.3445
0.5141  :  0.3602
0.5141  :  0.3739
0.5141  :  0.3844
```

```
0.5141  :  0.4074
=========================
0.5258  :  0.3134
0.5258  :  0.352
0.5258  :  0.3681
0.5258  :  0.3821
0.5258  :  0.393
0.5258  :  0.4165
=========================
0.5375  :  0.3199
0.5375  :  0.3595
0.5375  :  0.376
0.5375  :  0.3903
0.5375  :  0.4015
0.5375  :  0.4255
=========================
0.5492  :  0.3265
0.5492  :  0.367
0.5492  :  0.3841
0.5492  :  0.3986
0.5492  :  0.4101
0.5492  :  0.4348
=========================
0.5609  :  0.333
0.5609  :  0.3744
0.5609  :  0.3918
0.5609  :  0.4067
0.5609  :  0.4184
0.5609  :  0.4438
=========================
0.5725  :  0.34
0.5725  :  0.3824
0.5725  :  0.4002
0.5725  :  0.4153
0.5725  :  0.4273
0.5725  :  0.4533
=========================
0.5842  :  0.3465
0.5842  :  0.3898
0.5842  :  0.408
0.5842  :  0.4235
0.5842  :  0.4358
0.5842  :  0.4626
=========================
0.5959  :  0.3532
0.5959  :  0.3973
0.5959  :  0.416
0.5959  :  0.4318
```

```
0.5959  :  0.4444
0.5959  :  0.4718
=========================
0.6076  :  0.36
0.6076  :  0.405
0.6076  :  0.4241
0.6076  :  0.4402
0.6076  :  0.4531
0.6076  :  0.481
=========================
0.6193  :  0.3672
0.6193  :  0.413
0.6193  :  0.4325
0.6193  :  0.4489
0.6193  :  0.4621
0.6193  :  0.4906
=========================
0.631  :  0.374
0.631  :  0.4207
0.631  :  0.4406
0.631  :  0.4574
0.631  :  0.4709
0.631  :  0.5
=========================
0.6426  :  0.3811
0.6426  :  0.4287
0.6426  :  0.4489
0.6426  :  0.4661
0.6426  :  0.4798
0.6426  :  0.5094
=========================
0.6543  :  0.388
0.6543  :  0.4365
0.6543  :  0.4571
0.6543  :  0.4747
0.6543  :  0.4887
0.6543  :  0.5189
=========================
0.666  :  0.3949
0.666  :  0.4443
0.666  :  0.4652
0.666  :  0.4832
0.666  :  0.4975
0.666  :  0.5283
=========================
0.6777  :  0.4018
0.6777  :  0.4523
0.6777  :  0.4737
```

```
0.6777  :  0.4918
0.6777  :  0.5064
0.6777  :  0.5377
=========================
0.6894  :  0.4082
0.6894  :  0.4596
0.6894  :  0.4814
0.6894  :  0.5
0.6894  :  0.5149
0.6894  :  0.5469
=========================
0.7011  :  0.4151
0.7011  :  0.4675
0.7011  :  0.4896
0.7011  :  0.5086
0.7011  :  0.5238
0.7011  :  0.5564
=========================
0.7128  :  0.4219
0.7128  :  0.4753
0.7128  :  0.4978
0.7128  :  0.5172
0.7128  :  0.5328
0.7128  :  0.5659
=========================
0.7244  :  0.4291
0.7244  :  0.4834
0.7244  :  0.5063
0.7244  :  0.526
0.7244  :  0.5419
0.7244  :  0.5756
=========================
0.7361  :  0.4358
0.7361  :  0.491
0.7361  :  0.5143
0.7361  :  0.5344
0.7361  :  0.5507
0.7361  :  0.585
=========================
0.7478  :  0.4427
0.7478  :  0.4989
0.7478  :  0.5226
0.7478  :  0.543
0.7478  :  0.5597
0.7478  :  0.5946
=========================
0.7595  :  0.4497
0.7595  :  0.5068
```

```
0.7595  :  0.5308
0.7595  :  0.5516
0.7595  :  0.5687
0.7595  :  0.6041
=========================
0.7712  :  0.4566
0.7712  :  0.5145
0.7712  :  0.5389
0.7712  :  0.56
0.7712  :  0.5774
0.7712  :  0.6134
=========================
0.7829  :  0.4634
0.7829  :  0.5222
0.7829  :  0.5471
0.7829  :  0.5686
0.7829  :  0.5862
0.7829  :  0.6228
=========================
0.7945  :  0.4704
0.7945  :  0.5301
0.7945  :  0.5554
0.7945  :  0.5772
0.7945  :  0.5953
0.7945  :  0.6323
=========================
0.8062  :  0.4777
0.8062  :  0.5382
0.8062  :  0.5639
0.8062  :  0.586
0.8062  :  0.6044
0.8062  :  0.6419
=========================
0.8179  :  0.4846
0.8179  :  0.546
0.8179  :  0.5721
0.8179  :  0.5946
0.8179  :  0.6132
0.8179  :  0.6512
=========================
0.8296  :  0.4919
0.8296  :  0.5542
0.8296  :  0.5806
0.8296  :  0.6034
0.8296  :  0.6224
0.8296  :  0.6609
=========================
0.8413  :  0.4983
```

```
0.8413  :  0.5615
0.8413  :  0.5882
0.8413  :  0.6115
0.8413  :  0.6307
0.8413  :  0.6699
=========================
0.853  :  0.505
0.853  :  0.5691
0.853  :  0.5962
0.853  :  0.6198
0.853  :  0.6393
0.853  :  0.679
=========================
0.8646  :  0.5124
0.8646  :  0.5773
0.8646  :  0.6048
0.8646  :  0.6288
0.8646  :  0.6486
0.8646  :  0.6887
=========================
0.8763  :  0.5205
0.8763  :  0.5861
0.8763  :  0.6138
0.8763  :  0.638
0.8763  :  0.6582
0.8763  :  0.6988
=========================
0.888  :  0.5276
0.888  :  0.594
0.888  :  0.6222
0.888  :  0.6467
0.888  :  0.667
0.888  :  0.7082
=========================
0.8997  :  0.5353
0.8997  :  0.6025
0.8997  :  0.631
0.8997  :  0.6558
0.8997  :  0.6764
0.8997  :  0.718
=========================
0.9114  :  0.5427
0.9114  :  0.6108
0.9114  :  0.6396
0.9114  :  0.6647
0.9114  :  0.6855
0.9114  :  0.7276
=========================
```

```
0.9231  :  0.5503
0.9231  :  0.6191
0.9231  :  0.6483
0.9231  :  0.6736
0.9231  :  0.6947
0.9231  :  0.7374
=========================
0.9348  :  0.5582
0.9348  :  0.6278
0.9348  :  0.6572
0.9348  :  0.6827
0.9348  :  0.7041
0.9348  :  0.7473
=========================
0.9464  :  0.5657
0.9464  :  0.6358
0.9464  :  0.6655
0.9464  :  0.6913
0.9464  :  0.713
0.9464  :  0.7567
=========================
0.9581  :  0.5734
0.9581  :  0.6442
0.9581  :  0.6742
0.9581  :  0.7003
0.9581  :  0.7222
0.9581  :  0.7663
=========================
0.9698  :  0.5809
0.9698  :  0.6524
0.9698  :  0.6827
0.9698  :  0.7091
0.9698  :  0.7313
0.9698  :  0.7759
=========================
0.9815  :  0.5881
0.9815  :  0.6604
0.9815  :  0.6909
0.9815  :  0.7176
0.9815  :  0.7401
0.9815  :  0.7852
=========================
0.9932  :  0.5958
0.9932  :  0.6687
0.9932  :  0.6996
0.9932  :  0.7265
0.9932  :  0.7492
0.9932  :  0.7947
```

```
(0.6002059655984983, 0.6734663012888759, 0.7044471950024948, 0.7315881716225514, 0.75445961492
```

With threshold of **0.9**, I had the test accuracy of **0.8000891664837851**

```
In [32]: def predict_nn_3(model, input_vector, print_score = False):

             scores = model.predict(input_vector).reshape(100)
             predictions = np.where(scores > 0.9)[0]
             res = set(np.array(occupations)[predictions])
             return res
```

```
In [34]: from IPython.display import clear_output, display
         import time

         sequences_res = tokenizer.texts_to_sequences(summaries_test)
         data_res = pad_sequences(sequences_res, maxlen=maxlen)

         def export(start=0):
             with gzip.open('results.json.gz', 'wt') as output:
                 for i in range(start, len(titles_test)):
                     input_vector = data_res[i].reshape(1, -1)
                     prediction = predict_nn_3(model, input_vector)
                     sol = list(prediction)
                     output.write(json.dumps({'title':titles_test[i], 'prediction': sol}) + "\

                     clear_output(wait=True)
                     print(i,"/", len(titles_test), " - ", i * 100 / len(titles_test), "%")

         export()
```

```
643107 / 643108  -  99.999844505122 %
```

3 Summary

- Model Architecture: ***CNN + BiRNN***
- loss: ***weighted_binary_crossentropy*** - Give more weight to the 1-label
- Test Accuracy: ***0.8000891664837851***
- Threshold: ***0.9
- Using the most **20,000** common words in the dataset.
- Word Embedding: **Glove with 400,000 vocabs** (https://nlp.stanford.edu/projects/glove/).

3.1 Some Experiments to try to increase the accuracy but did not work

- I tested with other word embeddings like: word2vec, fasttext but the Glove still had the highest accuracy.
- I tried to increase and decrease the number of convolutional layers to 1 and 4 but the test accuracy dropped. 3 seems to be the optimal number.

- I also varied the kernel size of the convolutional and max_pooling layers but the test accuracy dropped.
- More dense layers and more RNN layers also did not help while the running time increase exponentially.
- However, the accuracy increase a lot when I increase the size of each RNN layer. Due to computational limitation I cannot test with bigger GRU layers.
- I increased the number of common words to 50,000 100,000 and 400,000 with bigger word vector vocab but the accuracy also dropped. This seems not intuitively because I think more words will result in better accuracy.
- I tried the jaccard_distance loss function but the model cannot learn.
- I tried to train the embedding layer but the result was not good.
- I also changed the number of words per summary to 100 and 500 but the accuracy is not better.

```
In [ ]:
```